Simulation

Lecture H3

Heuristic Methods: Genetic Algorithms

Saeed Bastani
saeed.bastani@eit.lth.se
Spring 2017

Outline

- ✓ Introduction to Genetic Algorithms (GA)
- ✓ How does GA works?
- ✓ GA Parameters and Operations
- ✓ Examples

Local Search Based Metaheuristics (1)

- The different methods employ very different techniques in order explore a larger part of the search space
  - Simulated Annealing relies on controlled random movement
  - Tabu Search relies on memory structures, recording enough information to guide the search to different areas of the search space (e.g., frequency based diversification)

Local Search Based Metaheuristics (2)

- Which method is better?
- Depends on your needs
  - SA is easier to implement?
  - SA is easier to use/understand?
  - TS is more flexible and robust?
  - TS requires a better understanding of the problem?
  - TS requires more ”tuning”?
  - TS produces better overall results?

Genetic Algorithms

- We have now studied two Metaheuristics based on the idea of a Local Search
- It is time to look at methods that are based on different mechanisms
- One such method will be the Genetic Algorithm

The Genetic Algorithm

- Directed search algorithms based on the mechanics of biological evolution
- Developed by John Holland, University of Michigan (1970's)
  - To understand the adaptive processes of natural systems
  - To design artificial systems software that retains the robustness of natural systems

Genetic Algorithms

- Provide efficient, effective techniques for optimization and machine learning applications
- Widely-used today in business, scientific and engineering circles

Genetic Algorithms (GA)

- Function Optimization
- AI (Games,Pattern recognition …)
- OR after a while
- Basic idea:
  - intelligent exploration of the search space based on random search
  - analogies from biology

GA - Analogies with biology

- Representation of complex objects by a vector of simple components
- Chromosomes
- Selective breeding
- Darwinistic evolution

- Classical GA: Binary encoding

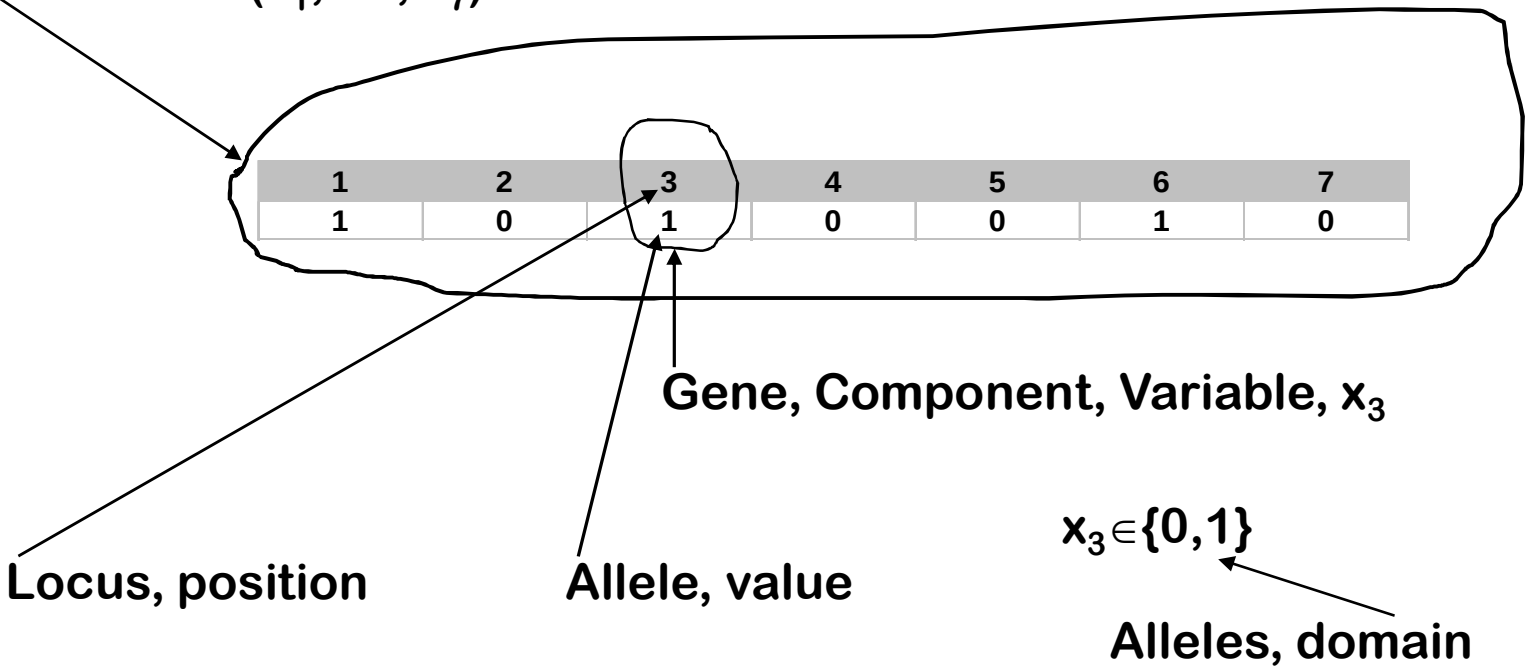
Components of a GA

A problem to solve, and ...

- Encoding technique (*gene, chromosome*)
- Initialization procedure *(creation)*
- Evaluation function *(environment)*
- Selection of parents *(reproduction)*
- Genetic operators *(mutation, recombination)*
- Parameter settings *(practice and art)*

Classical GA: Binary Chromosomes

Chromosome, component vector, vector, string, solution, individual $x=(x_1, \ldots, x_7)$



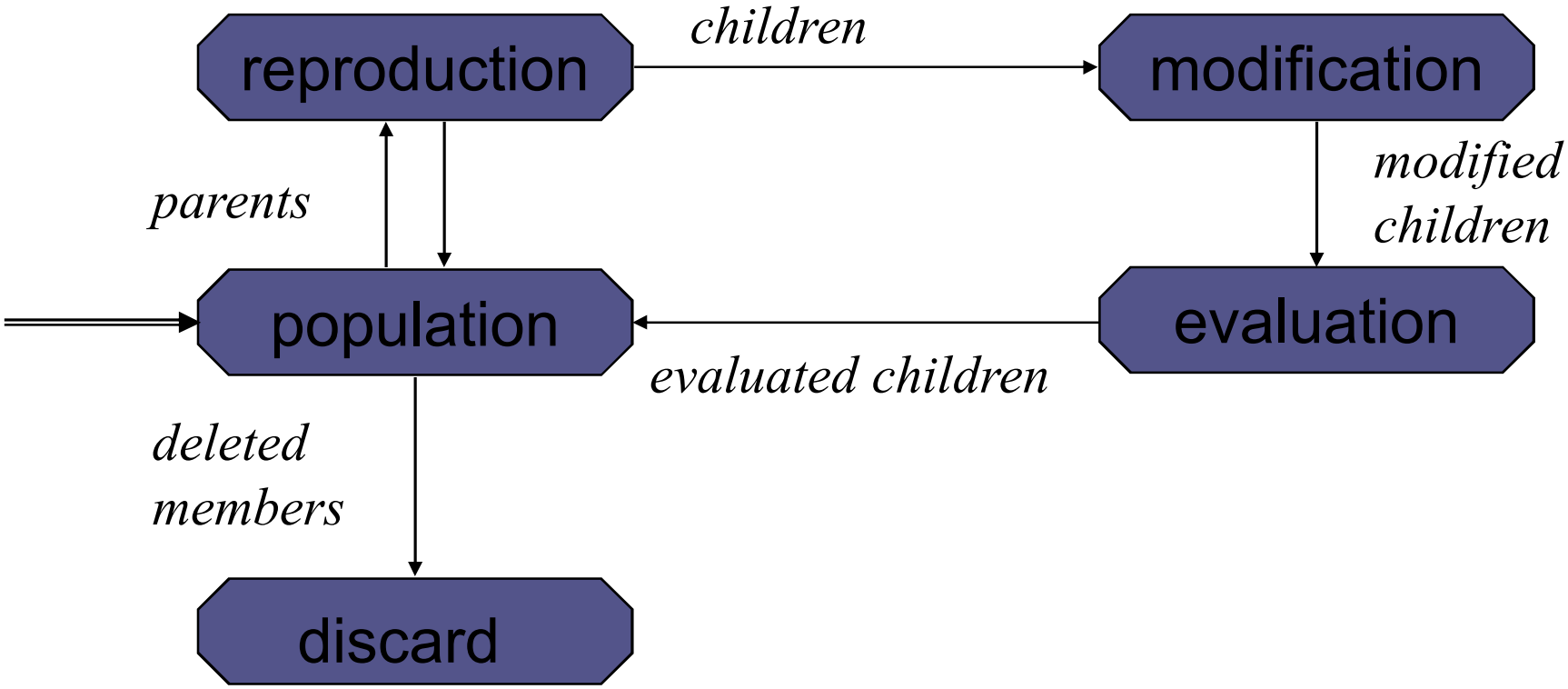
Genotype, Phenotype, Population

- Genotype
  - chromosome
  - Coding of chromosomes
  - coded string, set of coded strings
- Phenotype
  - The physical expression
  - Properties of a set of solutions
- Population – a set of solutions

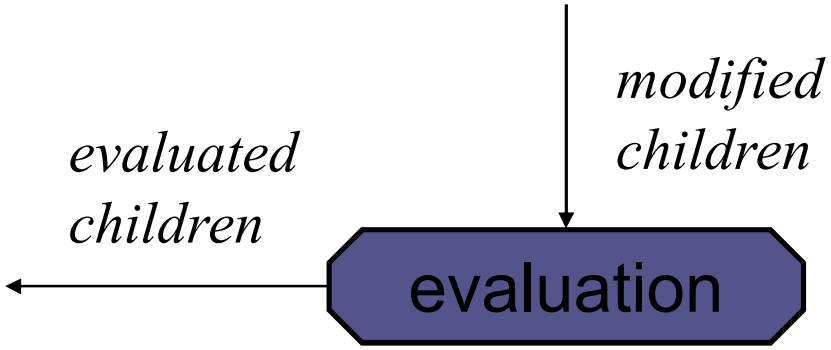
Genetic Algorithm

```
1: Choose an initial population of chromosomes
2: while stopping criterion not met do
3:    while sufficient offspring has not been created do
4:       if condition for crossover is satisfied then
5:          Select parent chromosomes
6:          Choose crossover parameters
7:          Perform crossover
8:       end if
9:       if condition for mutation is satisfied then
10:         Choose mutation points
11:         Perform mutation
12:      end if
13:      Evaluate fitness of offspring
14:   end while
15: end while
```

The GA Cycle of Reproduction



Evaluation



- The evaluator decodes a chromosome and assigns it a fitness measure
- The evaluator is the only link between a classical GA and the problem it is solving

Evaluation of Individuals

- Adaptability – ”fitness”
- Relates to the objective function value for a DOP
- Fitness is maximized
- Used in selection (”*Survival of the fittest*”)
- Often *normalized*

$$f : S \rightarrow [0,1]$$

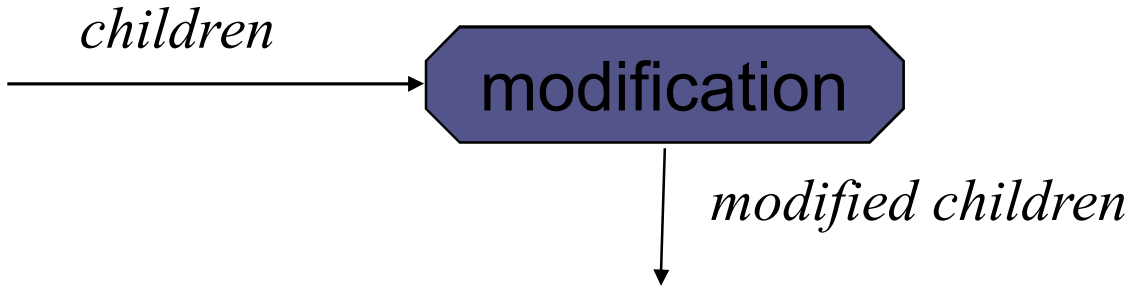
Genetic Operators

- Manipulates chromosomes/solutions
- Mutation: Unary operator
  - Inversions
- Crossover: Binary operator

GA - Evolution

- N generations of populations
- For every step in the evolution
  - Selection of individuals for genetic operations
  - Creation of new individuals (reproduction)
  - Mutation
  - Selection of individuals to survive
- Fixed population size M

Chromosome Modification



- Modifications are stochastically triggered
- Operator types are:
  - Mutation
  - Crossover (recombination)

GA - Mutation

1	2	3	4	5	6	7
1	0	1	0	0	1	0

1	2	3	4	5	6	7
1	0	1	1	0	1	0

Mutation: Local Modification

Before: (1 0 1 **1** 0 1 1 0)

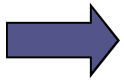
After: (1 0 1 **0** 0 1 1 0)

Before: (1.38 **-69.4** 326.44 0.1)

After: (1.38 **-67.5** 326.44 0.1)

- Causes movement in the search space (local or global)
- Restores lost information to the population

Crossover: Recombination



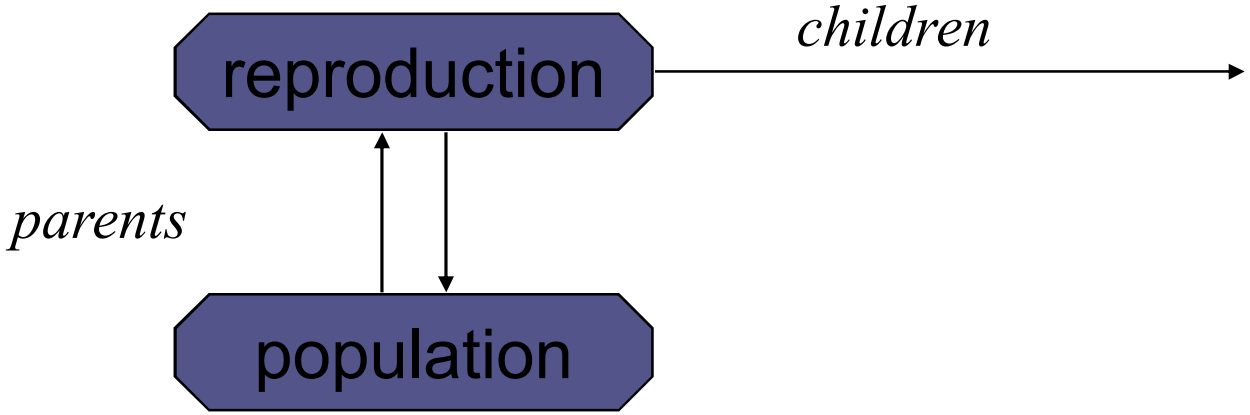
P1 (0 1 1 | 0 1 0 0 0) → (0 1 1 | 1 1 0 1 0) C1

P2 (1 1 0 | 1 1 0 1 0) → (1 1 0 | 0 1 0 0 0) C2

Crossover is a critical feature of genetic algorithms:

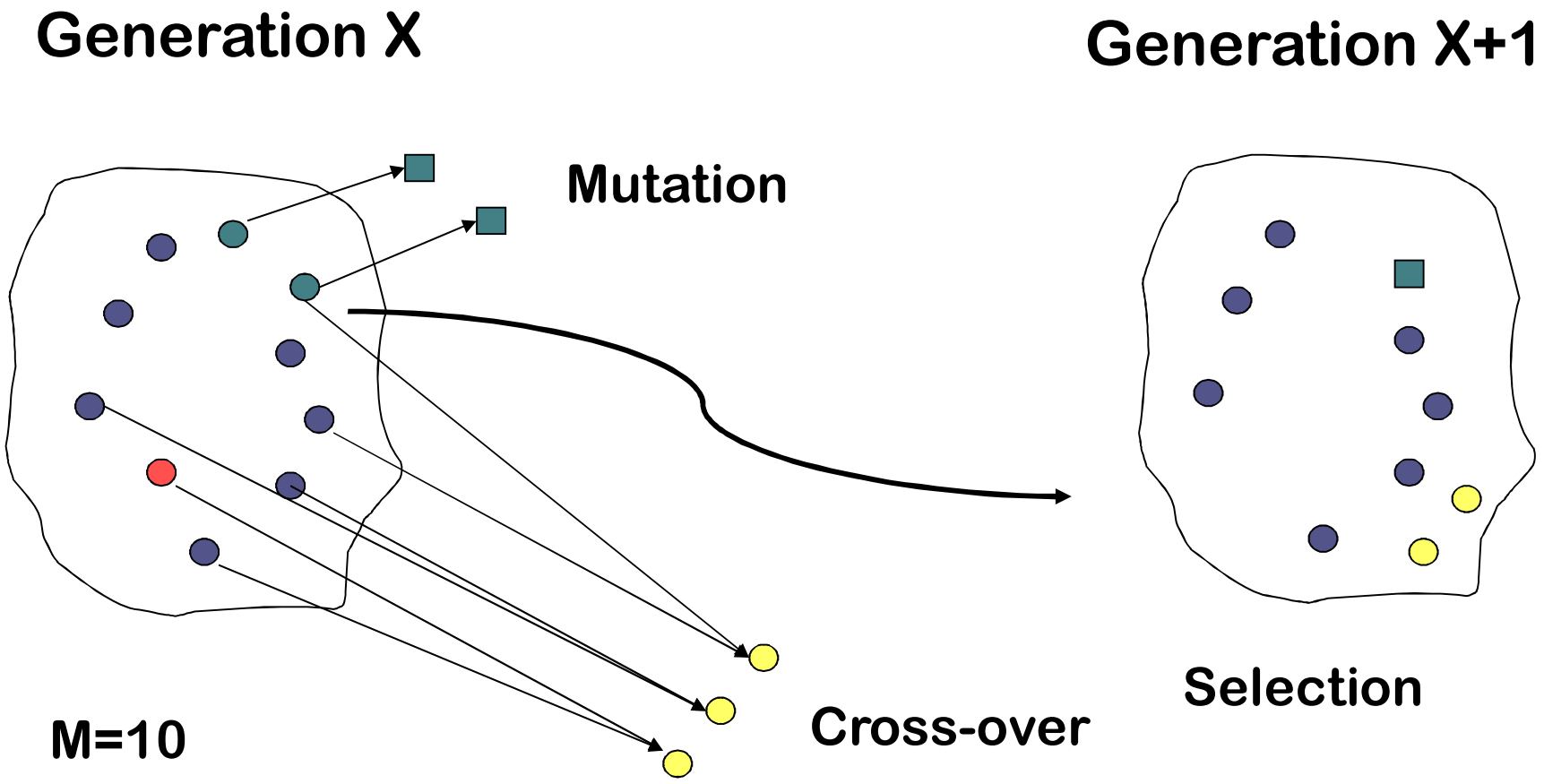
- It greatly accelerates search early in evolution of a population
- It leads to effective combination of schemata (subsolutions on different chromosomes)

Reproduction



Parents are selected at random with selection chances biased in relation to chromosome evaluations

GA - Evolution



Population

Chromosomes could be:

- Bit strings (0101 ... 1100)
- Real numbers (43.2 -33.1 ... 0.0 89.2)
- Permutations of element (E11 E3 E7 ... E1 E15)
- Lists of rules (R1 R2 R3 ... R22 R23)
- Program elements (genetic programming)
- ... any data structure ...

Classical GA: Binary chromosomes

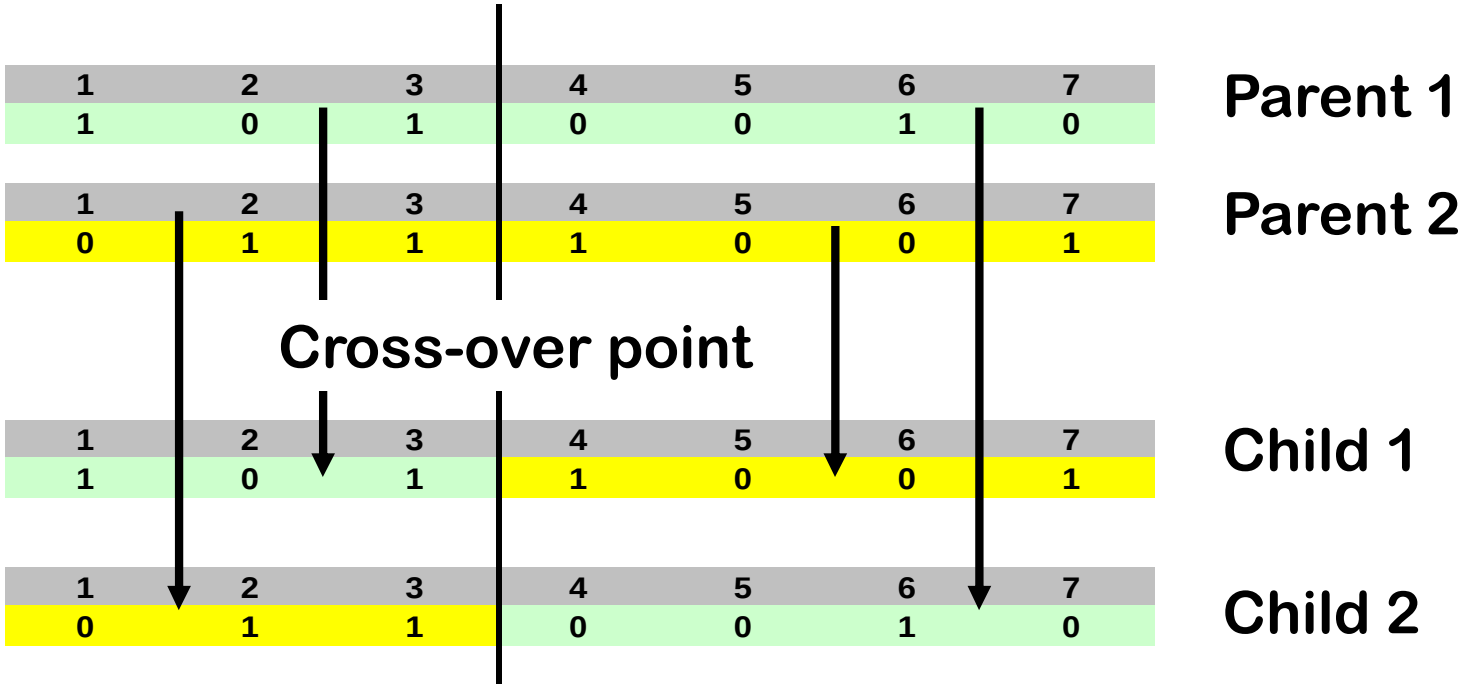
1	2	3	4	5	6	7
1	0	1	0	0	1	0

1	2	3	4	5	6	7
0	1	1	1	0	0	1

- Functional optimization
  - Chromosome corresponds to a binary encoding of a real number - min/max of an arbitrary function
- COP, TSP as an example
  - Binary encoding of a solution
  - Often better with a more direct representation (e.g. sequence representation)

GA - Classical Crossover (1-point)

- One parent is selected based on *fitness*
- The other parent is selected randomly
- Random choice of cross-over point



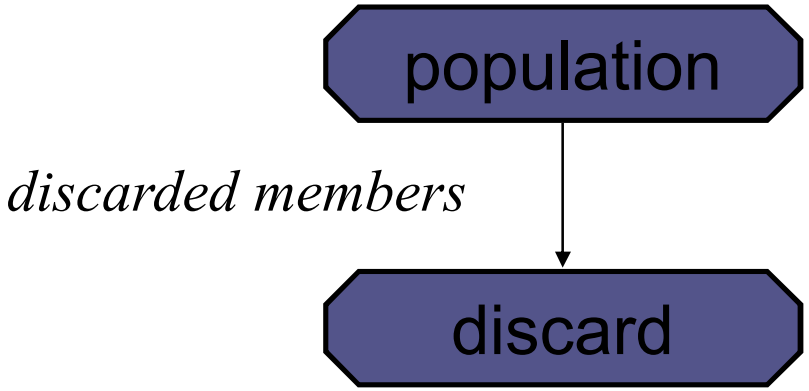
GA – Classical Crossover

- Arbitrary (or worst) individual in the population is changed with one of the two offspring (e.g. the best)
- Reproduce as long as you want
- Can be regarded as a sequence of almost equal populations
- Alternatively:
  - One parent selected according to fitness
  - Crossover until (at least) M offspring are created
  - The new population consists of the offspring
- Lots of other possibilities …
- Basic GA with classical crossover and mutation often works well

GA – Standard Reproduction Plan

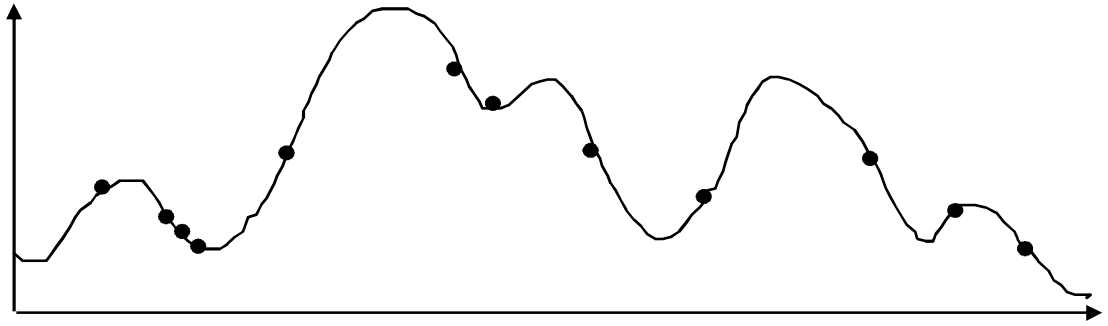
- Fixed population size
- Standard cross-over
  - One parent selected according to fitness
  - The other selected randomly
  - Random cross-over point
  - A random individual is exchanged with one of the offspring
- Mutation
  - A certain probability that an individual mutate
  - Random choice of which gene to mutate
  - Standard: mutation of offspring

Deletion

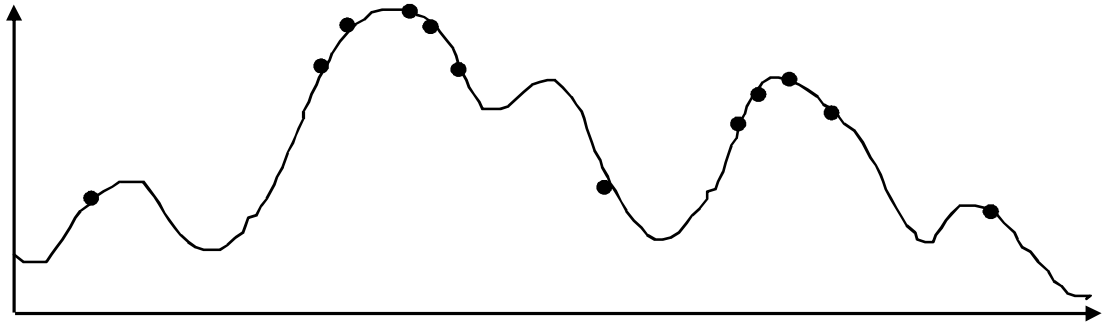


- *Generational* GA:
  entire populations replaced each iteration
- *Steady-state* GA:
  a few members replaced each generation

An Abstract Example



Distribution of Individuals in Generation 0



Distribution of Individuals in Generation N

A Simple Example

The Traveling Salesman Problem:

Find a tour of a given set of cities so that

- each city is visited only once
- the total distance traveled is minimized

Representation

Representation is an ordered list of city numbers known as an *order-based* GA.

1) London	3) Dunedin	5) Beijing	7) Tokyo
2) Venice	4) Singapore	6) Phoenix	8) Victoria

City List 1 (3 5 7 2 1 6 4 8)

City List 2 (2 5 7 6 8 1 3 4)

Crossover

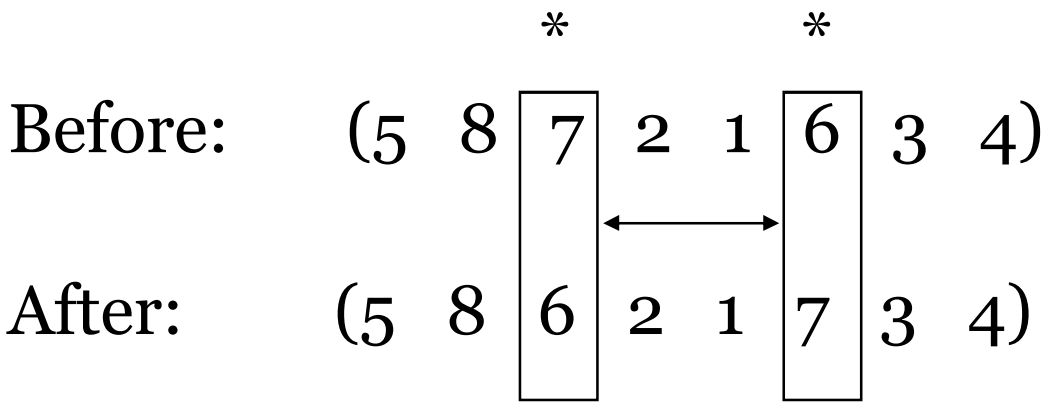
Crossover combines inversion and recombination:

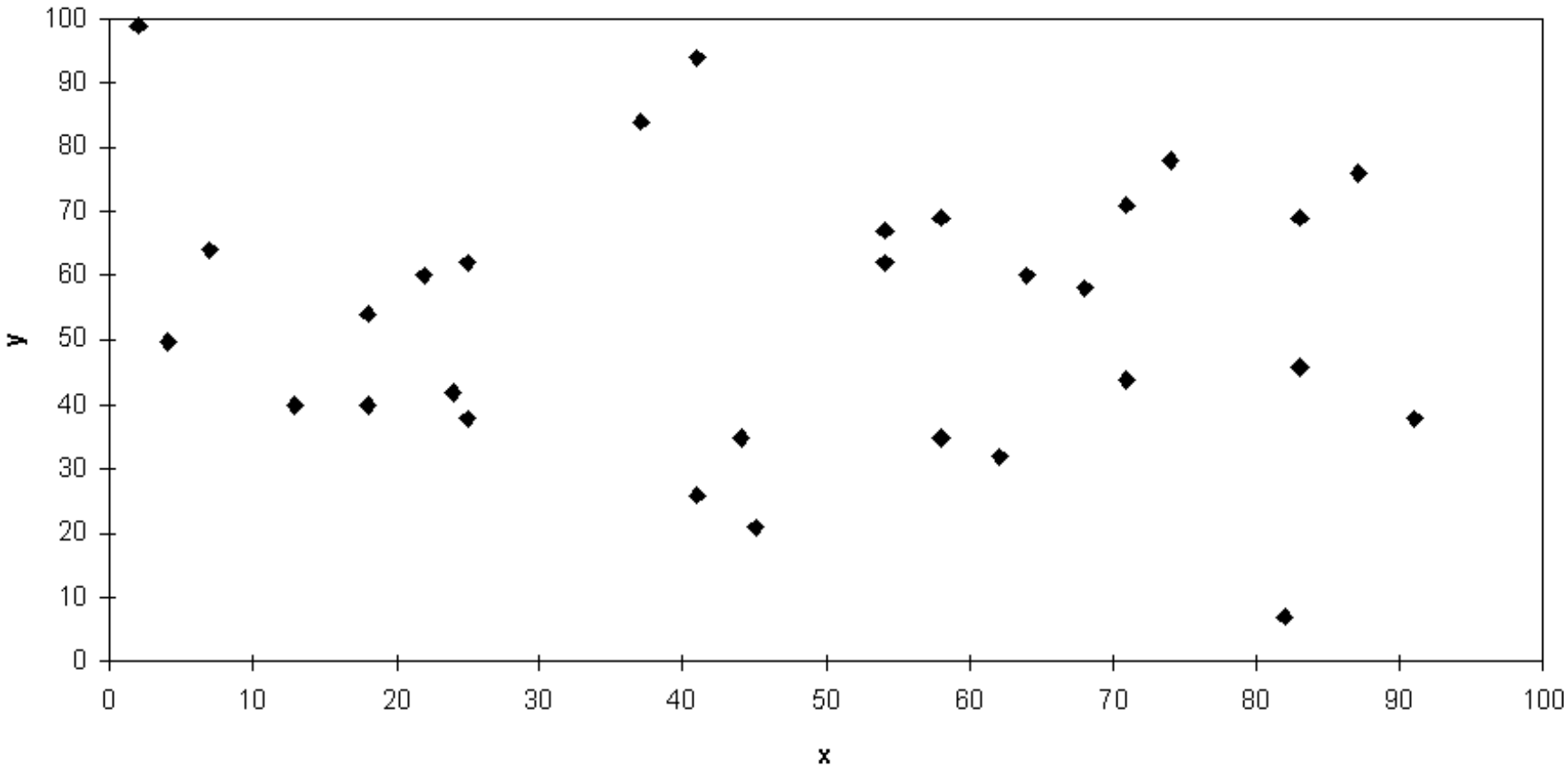
Parent1	(3 5 **7 2 1 6** 4 8)
Parent2	(2 5 7 6 8 1 3 4)
Child	(5 8 **7 2 1 6** 3 4)

This operator is called order-based crossover.

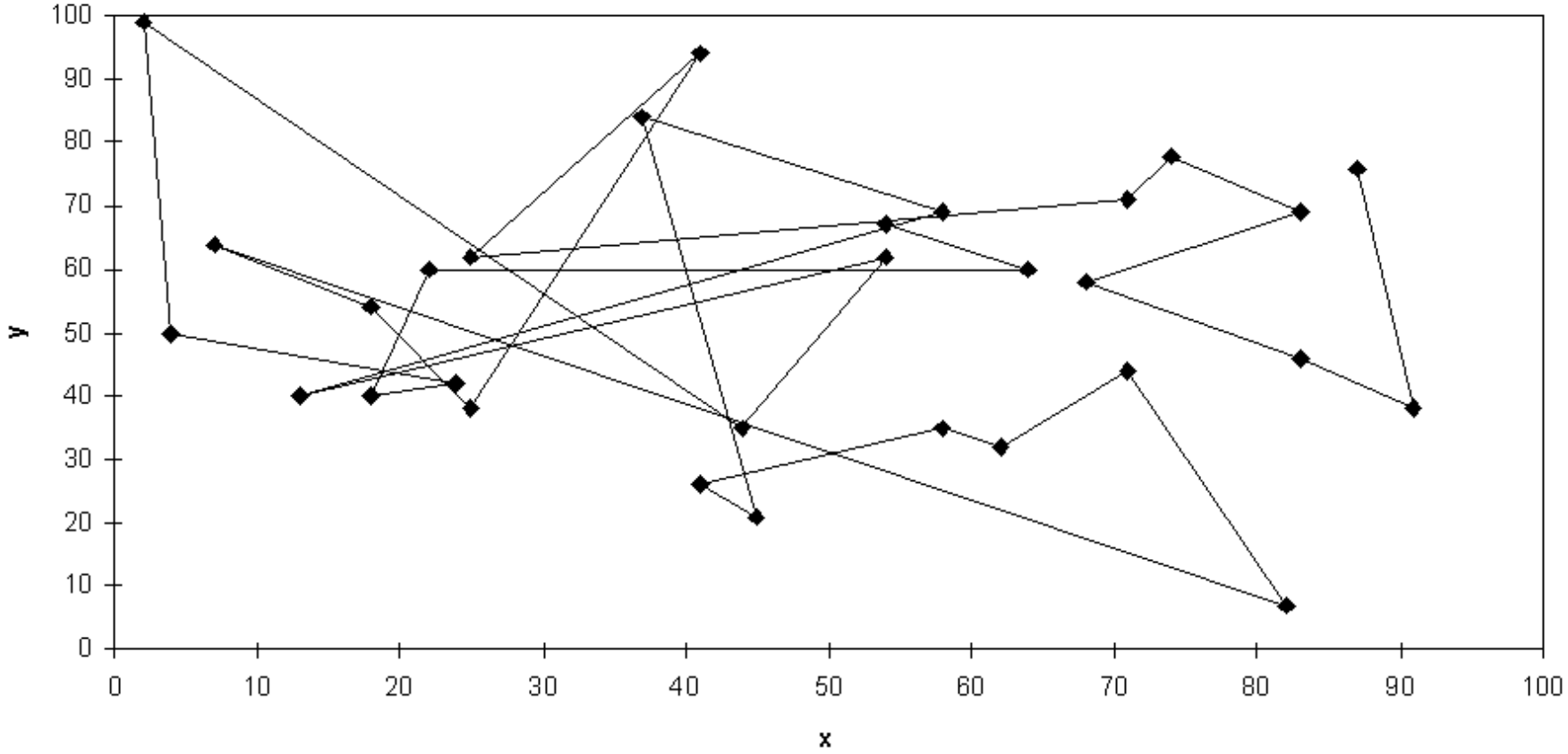
Mutation

Mutation involves reordering of the list:

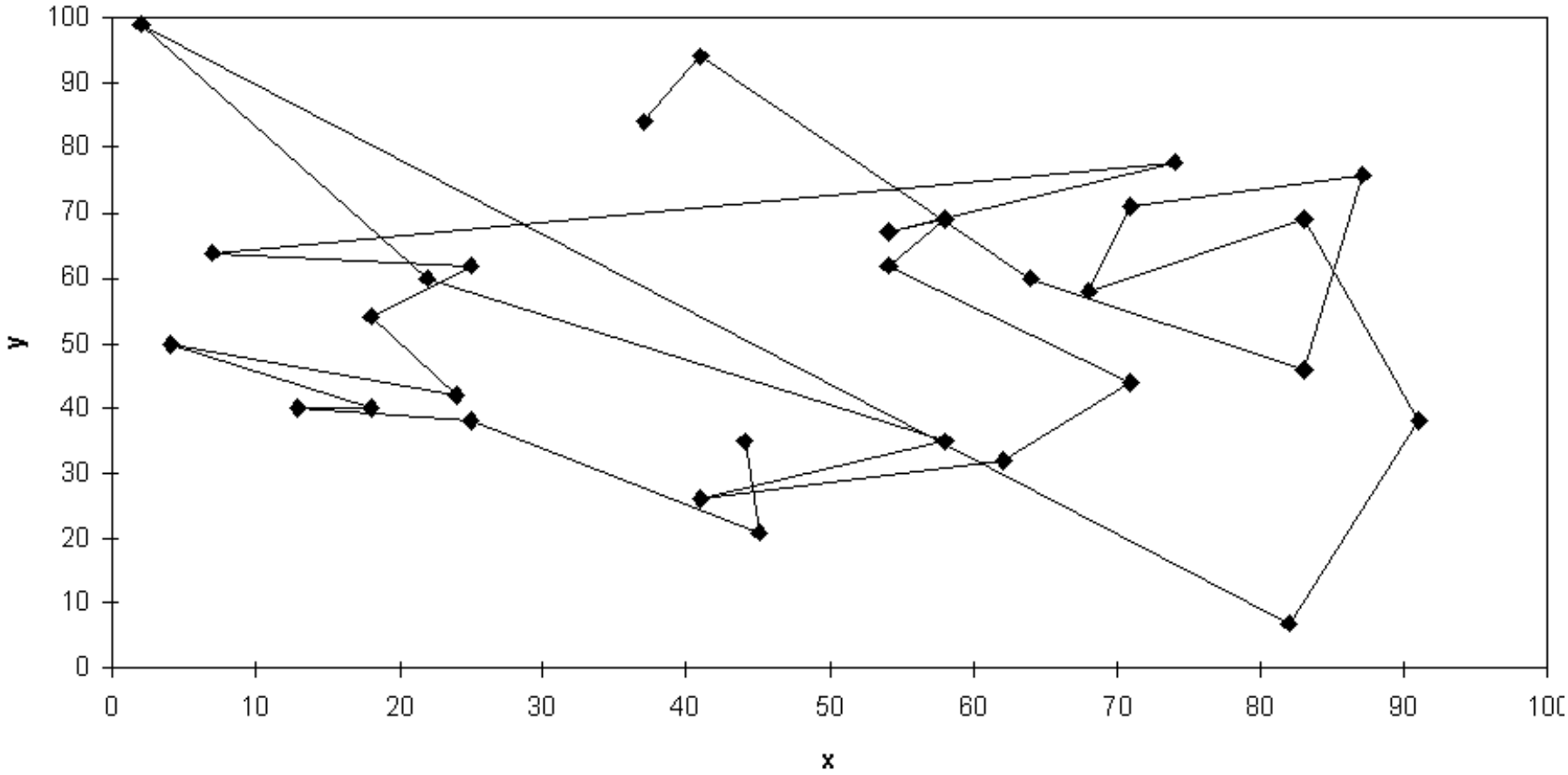



100
90
80
70
60
50
40
30
20
10
0
y
0
10
20
30
40
50
60
70
80
90
100
x

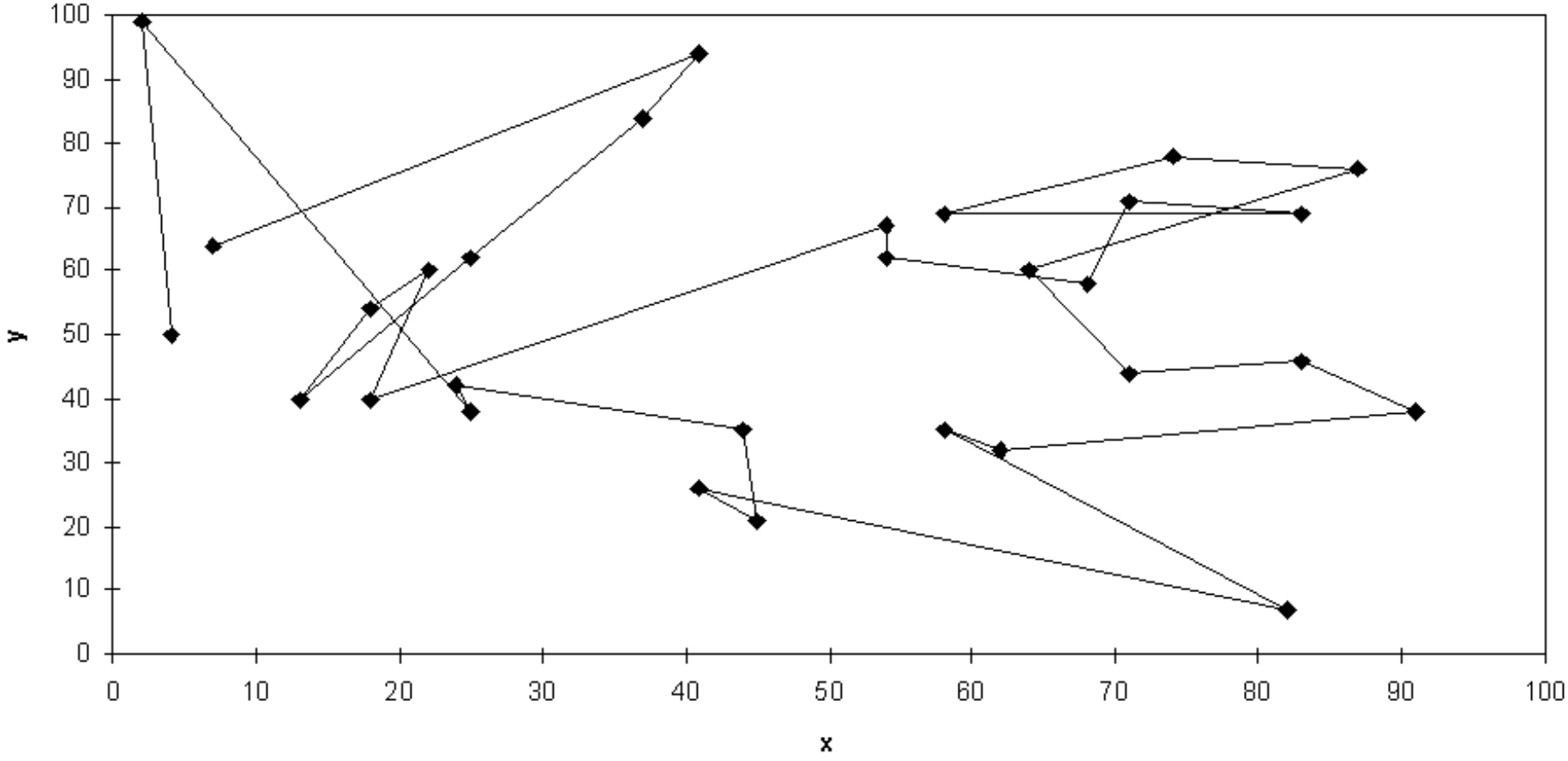
Solution i (Distance = 941)



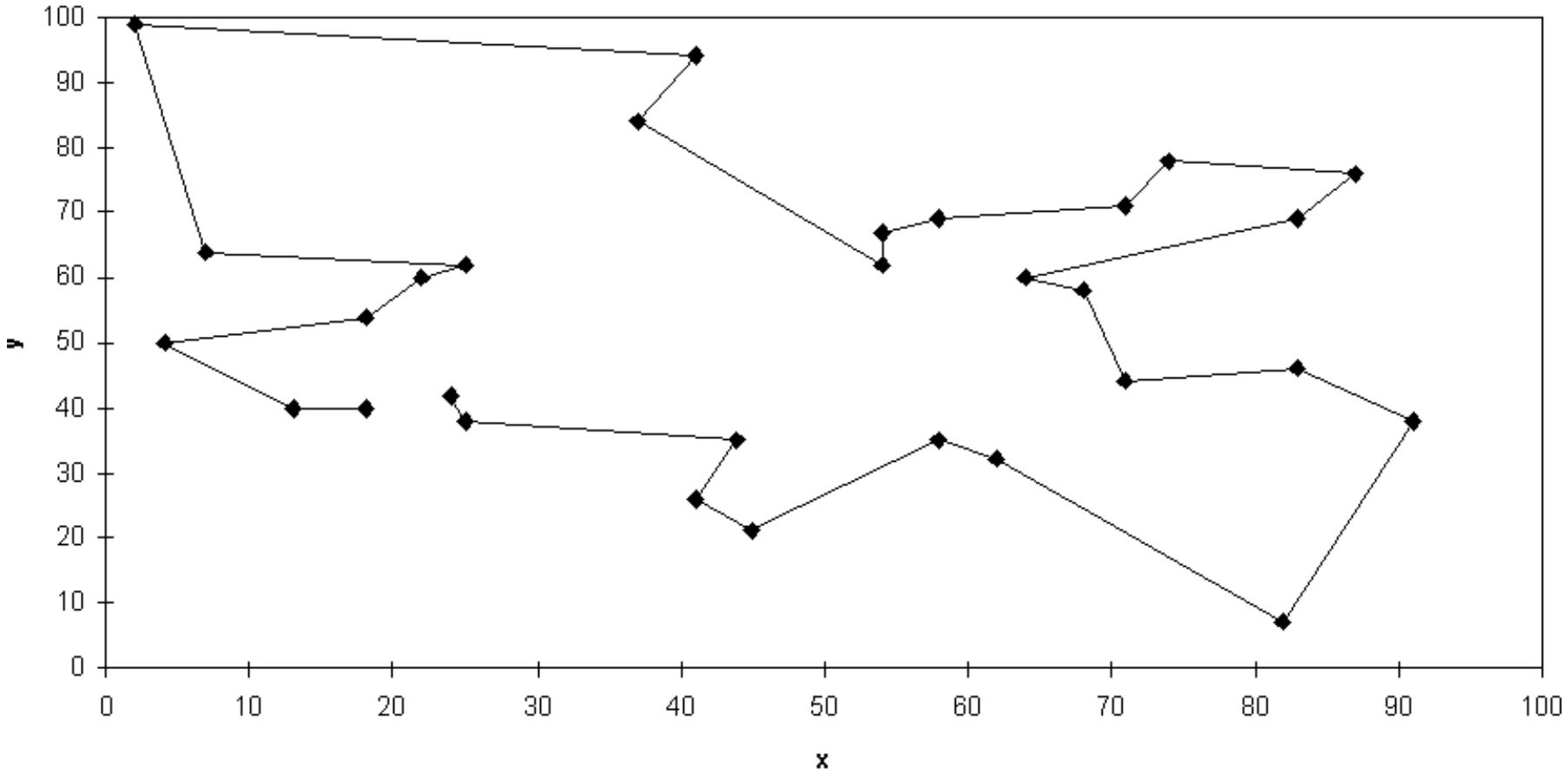
Solution j (Distance = 800)

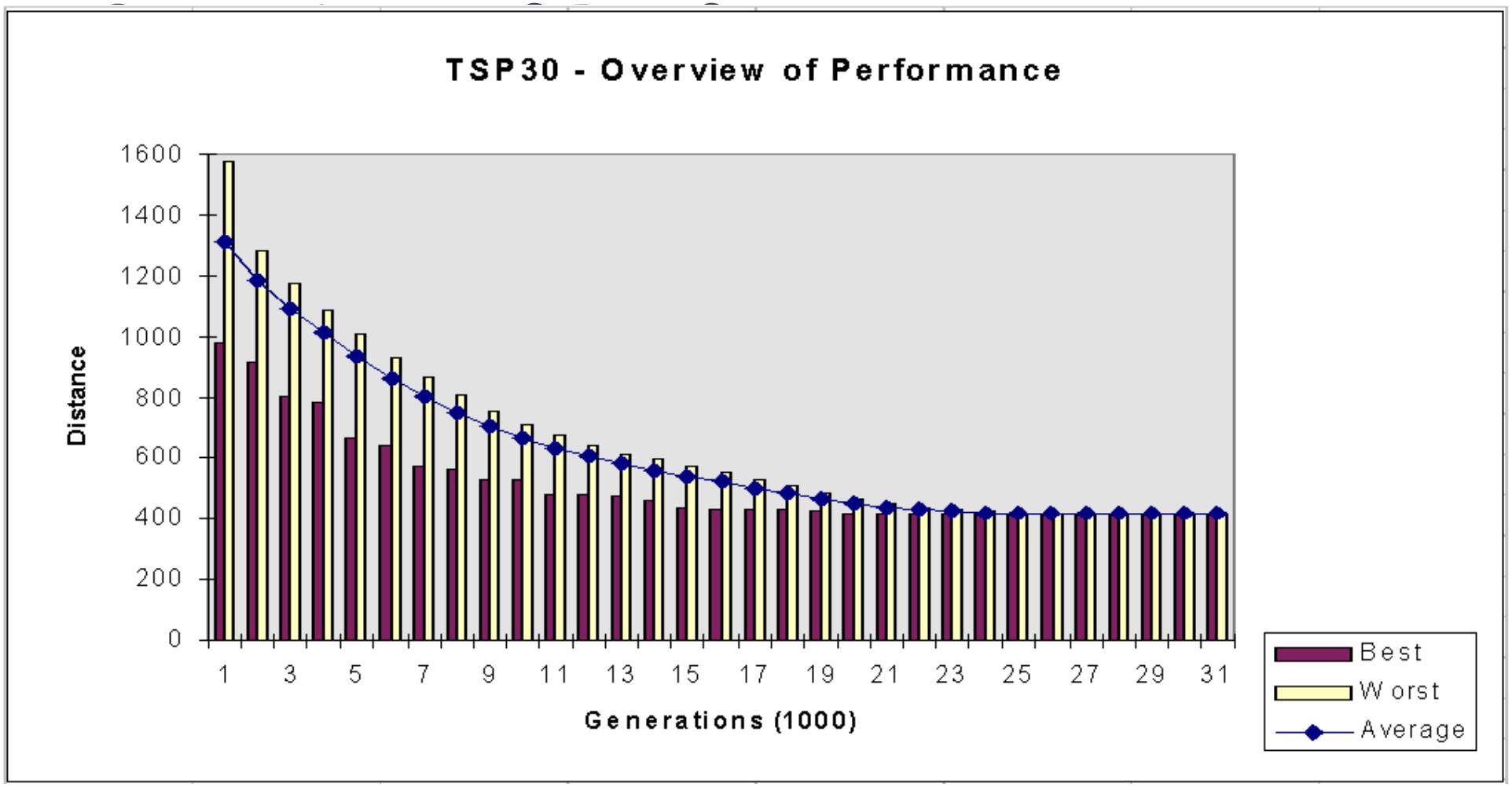


Solution k (Distance = 652)



Best Solution (Distance = 420)




TSP30 - Overview of Performance
Distance
1600
1400
1200
1000
800
600
400
200
0
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
31
Generations (1000)
Best
Worst
Average

An Example with Binary Coding Representation

Initial Population and Coding

Consider the problem: $\max_{\mathbf{x}} f(\mathbf{x}),\ \mathbf{x} \in \Re^n$

$$a_i \le x_i \le b_i, i = 1,2,...,n$$

Suppose we wish to represent x_i to d decimal places. That is each range $a_i \to b_i$ needs to be cut into $(b_i - a_i).10^d$ equal sizes. Let m_i be the smallest integer such that $(b_i - a_i) * 10^d \le 2^{m_i} - 1.$

Then x_i can be coded as a binary string of length m_i. Also, to interpret the string, we use:

$$x_i = a_i + decimal('\text{binary string}') * \frac{b_i - a_i}{2^{m_i} - 1}$$

Each chromosome (population member) is represented by a binary string of length:

$$m = \sum_{i=1}^{n} m_i$$

where the first m_1 bits map x_1 into a value from the range $[a_1, b_1]$, the next group of m_2 bits map x_2 into a value from the range $[a_2, b_2]$ etc; the last m_n bits map x_n into a value from the range $[a_n, b_n]$.

To initialise a population, we need to decide upon the number of chromosomes (*pop_size*). We then initialise the bit patterns, often randomly, to provide an initial set of potential solutions.

Selection (*roulette wheel principle*)

We mathematically construct a '*roulette wheel*' with slots sized according to fitness values. Spinning this wheel will then select a new population according to these fitness values with the chromosomes with the highest fitness having the greatest chance of selection (see the procedure in next slide)

1) Calculate the fitness value eval(v_i) for each chromosome v_i ($i = 1,...,pop\_size$)

2) Find the total fitness of the population

$$F = \sum_{i=1}^{pop\_size} eval(v_i)$$

3) Calculate the probability of a selection, p_i, for each chromosome v_i ($i = 1,...,pop\_size$)

$$p_i = \frac{eval(v_i)}{F}$$

4) Calculate a cumulative probability q_i for each chromosome v_i ($i = 1,...,pop\_size$)

$$q_i = \sum_{j=1}^{i} p_i$$

The selection process:

- spin the roulette wheel *pop_size* times and choose a chromosome each time
- How? (see below)

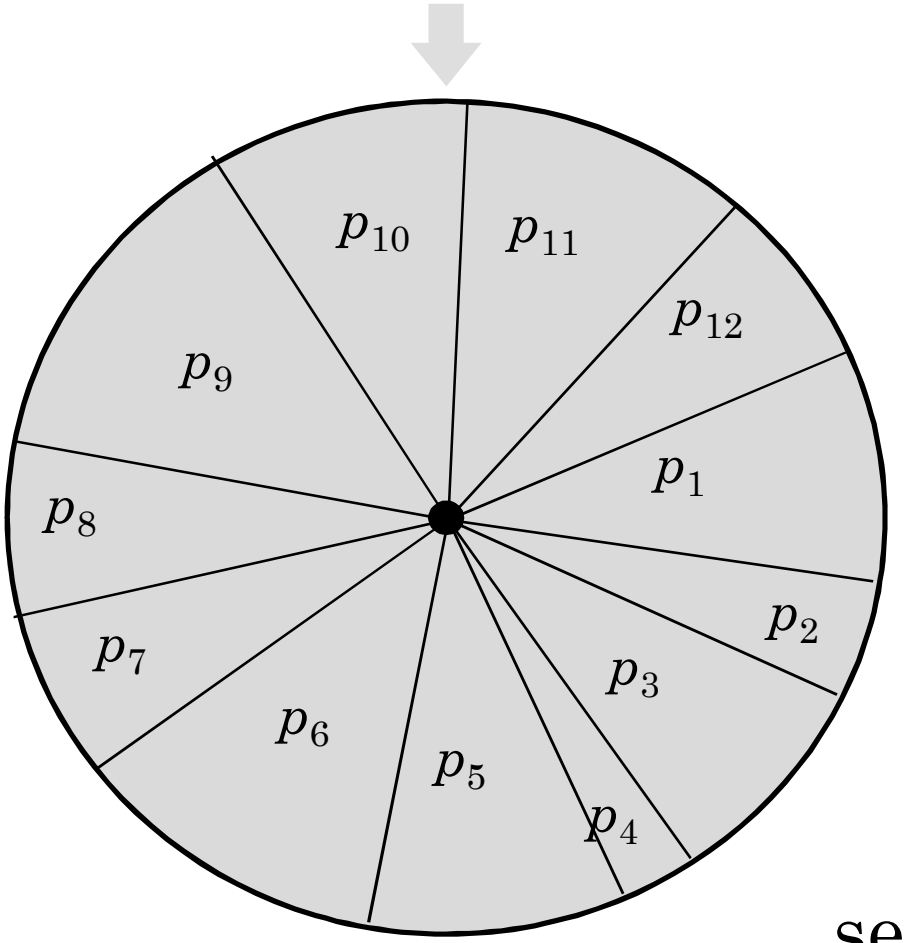
1) Generate a random number r in the range [0,1]
2) If $r < q_1$, select the first chromosome v_1; otherwise select the i^{th} chromosome v_i such that:

$$q_{i-1} < r \leq q_i \ , \ (2 \leq i \leq pop\_size)$$

Note that some chromosomes would be selected more than once: the best chromosomes get more copies and worst die off → ***"survival of the fittest"***

All the chromosomes selected then replace the previous set to obtain a new population.

Example:



segment area proportional to p_i, i=1,...,12

Crossover

We choose a parameter value p_c as the probability of crossover. Then the expected number of chromosomes to undergo the crossover operation will be p_c*pop_size. We proceed as follows.

For each chromosome in the new population do:

1) Generate a random number r from the range [0,1].

2) If $r < p_c$, then select the given chromosome for crossover.

ensuring that an even number is selected. Now we *mate* the selected chromosomes randomly.

For each pair of chromosomes we generate a random number *pos* from the range [1,*m*-1], where *m* is the number of bits in each chromosome. The number *pos* indicates the position of the crossing point. Two chromosomes:

$$\begin{array}{llllll} (b_1 & b_2 & \ldots & b_{pos} & b_{pos+1} & \ldots\ b_m) \\ (c_1 & c_2 & \ldots & c_{pos} & c_{pos+1} & \ldots\ c_m) \end{array}$$

are replaced by a pair of their offspring (*children*)

$$\begin{array}{llllll} (b_1 & b_2 & \ldots & b_{pos} & c_{pos+1} & \ldots\ c_m) \\ (c_1 & c_2 & \ldots & c_{pos} & b_{pos+1} & \ldots\ b_m) \end{array}$$

Mutation

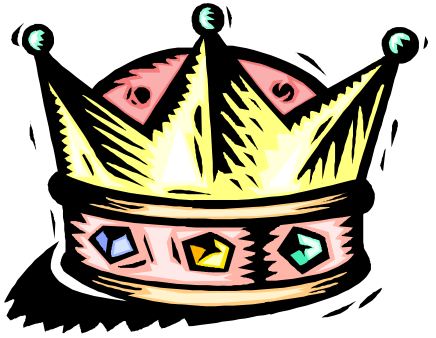
We choose a parameter value p_m as the probability of mutation. Mutation is performed on a bit-by-bit basis giving the expected number of mutated bits as $p_m * m * pop\_size$. Every bit, in all chromosomes in the whole population, has an equal chance to undergo mutation, that is change from a 0 to 1 or vice versa. The procedure is:

For each chromosome in the current population, and for each bit within the chromosome:

1) Generate a random number r from the range [0,1].

2) If $r < p_m$, mutate the bit.

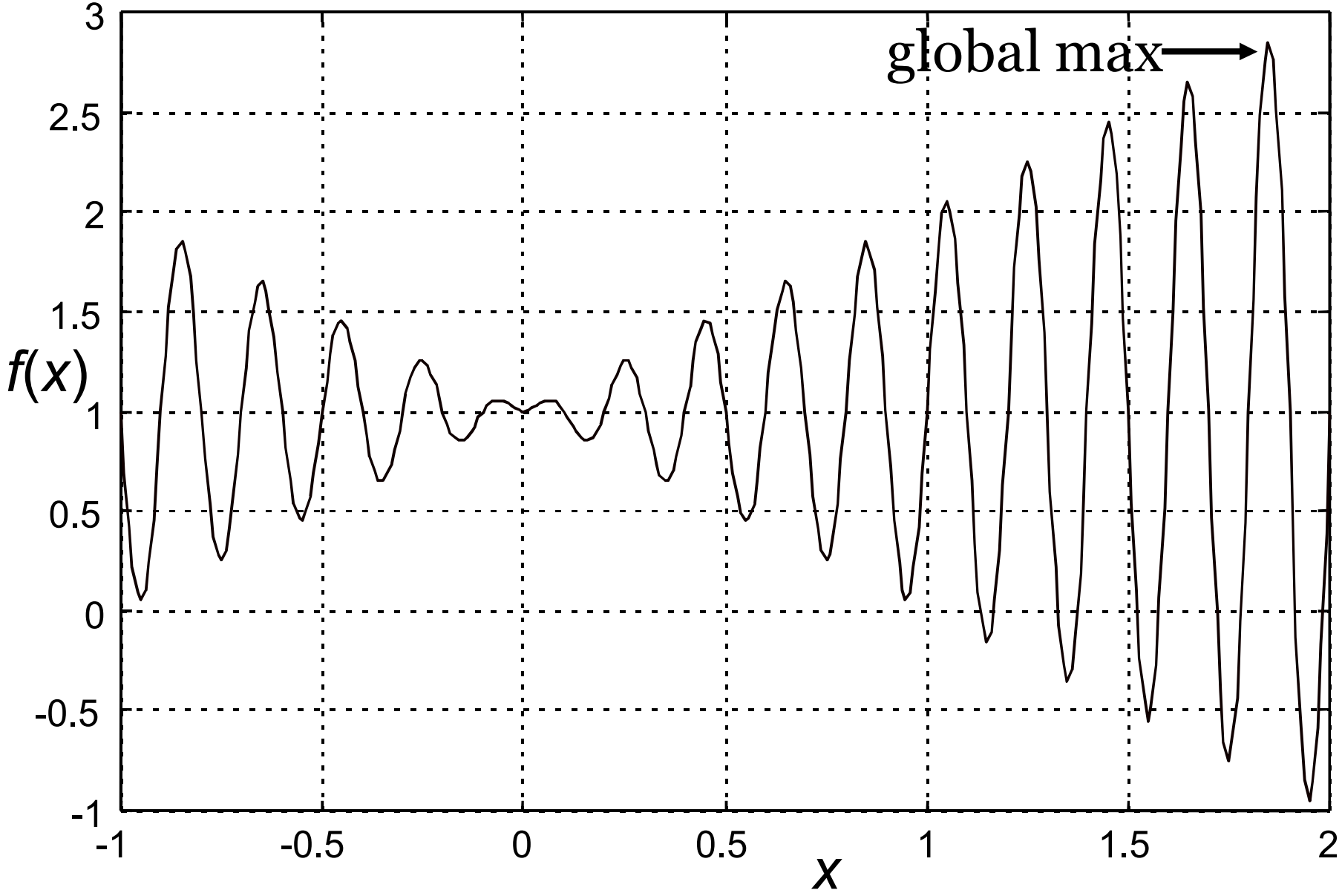
Elitism



It is usual to have a means for ensuring that the best value in a population is not lost in the selection process. One way is to store the best value before selection and, after selection, replace the poorest value with this stored best value.

Example

$$\max_x f(x) = x \sin(10\pi x) + 1.0, \quad -1 \le x \le 2$$



Representation

Let us work to a precision of two decimal places. then the chromosome length m must satisfy:

$$[2-(-1)]10^2 \leq 2^m - 1 \rightarrow 2^m \geq 301 \rightarrow m = 9$$

Also let $pop\_size = 10$, $p_c = 0.25$, $p_m = 0.04$

To ensure that a positive fitness value is always achieved we will work on $val = f(x) + 2$

Consider that the initial population has been randomly selected as follows (giving also the corresponding values of *x*, *val*, probabilities and accumulated probabilities)

	population	*x*	*val*	*p*	*q*
v_1	0 1 1 1 0 1 1 0 1	0.39	2.89	0.09	0.09
v_2	0 0 0 1 1 1 0 1 0	-0.66	3.63	0.11	0.20
v_3*	1 1 0 1 0 0 0 1 0	1.45	4.44	0.14	0.34
v_4	1 0 1 0 1 1 1 1 0	1.05	4.04	0.13	0.47
v_5	0 0 1 0 0 1 1 0 1	-0.55	2.45	0.08	0.55
v_6	0 0 0 1 0 0 1 1 0	-0.78	2.48	0.08	0.63
v_7	0 0 0 0 0 0 1 1 1 0	-0.92	2.51	0.08	0.71
v_8	0 0 0 1 1 1 0 0 0	-0.67	3.53	0.11	0.82
v_9	0 1 1 1 1 1 1 1 1 1	0.50	3.05	0.09	0.91
v_{10}	1 0 0 1 1 0 0 1 1	0.80	3.06	0.09	1.00

* fittest member of the population

Note for v_1: $dec(v_1) = 1 + 2^2 + 2^3 + 2^5 + 2^6 + 2^7 = 237$

$$\rightarrow x = -1 + \frac{237 \times 3}{2^9 - 1} = 0.39$$

$$F = \sum val = 32.08 \rightarrow p = \frac{2.89}{32.08} = 0.09$$

Selection

Assume 10 random numbers, range [0,1], have been obtained as follows:

0.47 0.61 0.72 0.03 0.18 0.69 0.83 0.68 0.54 0.83

These will select:

0.47 0.61 0.72 0.03 0.18 0.69 0.83 0.68 0.54 0.83

v_4 v_6 v_8 v_1 v_2 v_7 v_9 v_7 v_5 v_9

giving the new population:

	Population before selection	selection	Population after selection
v_1	0 1 1 1 0 1 1 0 1	4	1 0 1 0 1 1 1 1 0
v_2	0 0 0 1 1 1 0 1 0	5	0 0 0 1 0 0 1 1 0
v_3	1 1 0 1 0 0 0 1 0		0 0 0 1 1 1 0 0 0
v_4	1 0 1 0 1 1 1 1 0	1	0 1 1 1 0 1 1 0 1
v_5	0 0 1 0 0 1 1 0 1	9	0 0 0 1 1 1 0 1 0
v_6	0 0 0 1 0 0 1 1 0	2	0 0 0 0 0 1 1 1 0
v_7	0 0 0 0 0 1 1 1 0	6,8	0 1 1 1 1 1 1 1 1
v_8	0 0 0 1 1 1 0 0 0	3	0 0 0 0 0 1 1 1 0
v_9	0 1 1 1 1 1 1 1 1	7,10	0 0 1 0 0 1 1 0 1
v_{10}	1 0 0 1 1 0 0 1 1		0 1 1 1 1 1 1 1 1

Note that the best chromosome v_3 in the original population has not been selected and would be destroyed unless elitism is applied.

Crossover ($p_c = 0.25$)

Assume the 10 random numbers:

1	2	3	4	5	6	7	8	9	10
0.07	0.94	0.57	0.36	0.31	**0.14**	0.60	**0.07**	**0.07**	1.00

These will select v_1, v_6, v_8, v_9 for crossover.

Now assume 2 more random numbers in the range [1,8] are obtained:

7.20 3.35 $\rightarrow$ bits 8 and 4.

Mating v_1 and v_6 crossing over at bit 8:

v_1	1	0	1	0	1	1	1	1	0
v_6	0	0	0	0	0	1	1	1	0

⟶ *no change*

Mating v_8 and v_9 crossing over at bit 4:

v_8	0	0	0	0	0	1	1	1	0
v_9	0	0	1	0	0	1	1	0	1

↓ *produces*

v_8	0	0	0	0	0	1	1	0	1
v_9	0	0	1	0	0	1	1	1	0

giving the new population:

	population before crossover	population after crossover
v_1	1 0 1 0 1 1 1 1 0	1 0 1 0 1 1 1 1 0
v_2	0 0 0 1 0 0 1 1 0	0 0 0 1 0 0 1 1 0
v_3	0 0 0 1 1 1 0 0 0	0 0 0 1 1 1 0 0 0
v_4	0 1 1 1 0 1 1 0 1	0 1 1 1 0 1 1 0 1
v_5	0 0 0 1 1 1 0 1 0	0 0 0 1 1 1 0 1 0
v_6	0 0 0 0 0 1 1 1 0	0 0 0 0 0 1 1 1 0
v_7	0 1 1 1 1 1 1 1 1	0 1 1 1 1 1 1 1 1
v_8	0 0 0 0 0 1 1 1 0	0 0 0 0 0 1 1 0 1
v_9	0 0 1 0 0 1 1 0 1	0 0 1 0 0 1 1 1 0
v_{10}	0 1 1 1 1 1 1 1 1	0 1 1 1 1 1 1 1 1

⟶ bit for mutation

Mutation ($p_m = 0.04$)

Suppose a random number generator selects bit 2 of v_2 and bit 8 of v_9 to mutate, resulting in:

	population after mutation	x	val
v_1	1 0 1 0 1 1 1 1 0	1.05	4.04
v_2	0 1 0 1 0 0 1 1 0	-0.02	3.02
v_3	0 0 0 1 1 1 0 0 0	-0.67	3.53
v_4	0 1 1 1 0 1 1 0 1	0.39	2.89
v_5	0 0 0 1 1 1 0 1 0	-0.66	3.63
v_6	0 0 0 0 0 1 1 1 0	-0.92	2.51
v_7	0 1 1 1 1 1 1 1 1	0.50	3.05
v_8**	0 0 0 0 0 1 1 0 1	-0.92	2.37
v_9	0 0 1 0 0 1 1 0 0	-0.55	2.45
v_{10}	0 1 1 1 1 1 1 1 1	0.50	3.05

Total fitness $F = \underline{30.54}$

** weakest

Elitism

So far the iteration has resulted in a decrease in overall fitness (from 32.08 to 30.54). However, if we now apply elitism we replace v_8 in the current population by v_3 from the original population, to produce:

	population after mutation	x	val
v_1	1 0 1 0 1 1 1 1 1 0	1.05	4.04
v_2	0 1 0 1 0 0 1 1 1 0	-0.02	3.02
v_3	0 0 0 1 1 1 1 0 0 0	-0.67	3.53
v_4	0 1 1 1 0 1 1 0 1	0.39	2.89
v_5	0 0 0 1 1 1 1 0 1 0	-0.66	3.63
v_6	0 0 0 0 0 0 1 1 1 0	-0.92	2.51
v_7	0 1 1 1 1 1 1 1 1 1	0.50	3.05
v_8	1 1 0 1 0 0 0 1 0	1.45	4.44
v_9	0 0 1 0 0 1 1 0 0	-0.55	2.45
v_{10}	0 1 1 1 1 1 1 1 1 1	0.50	3.05

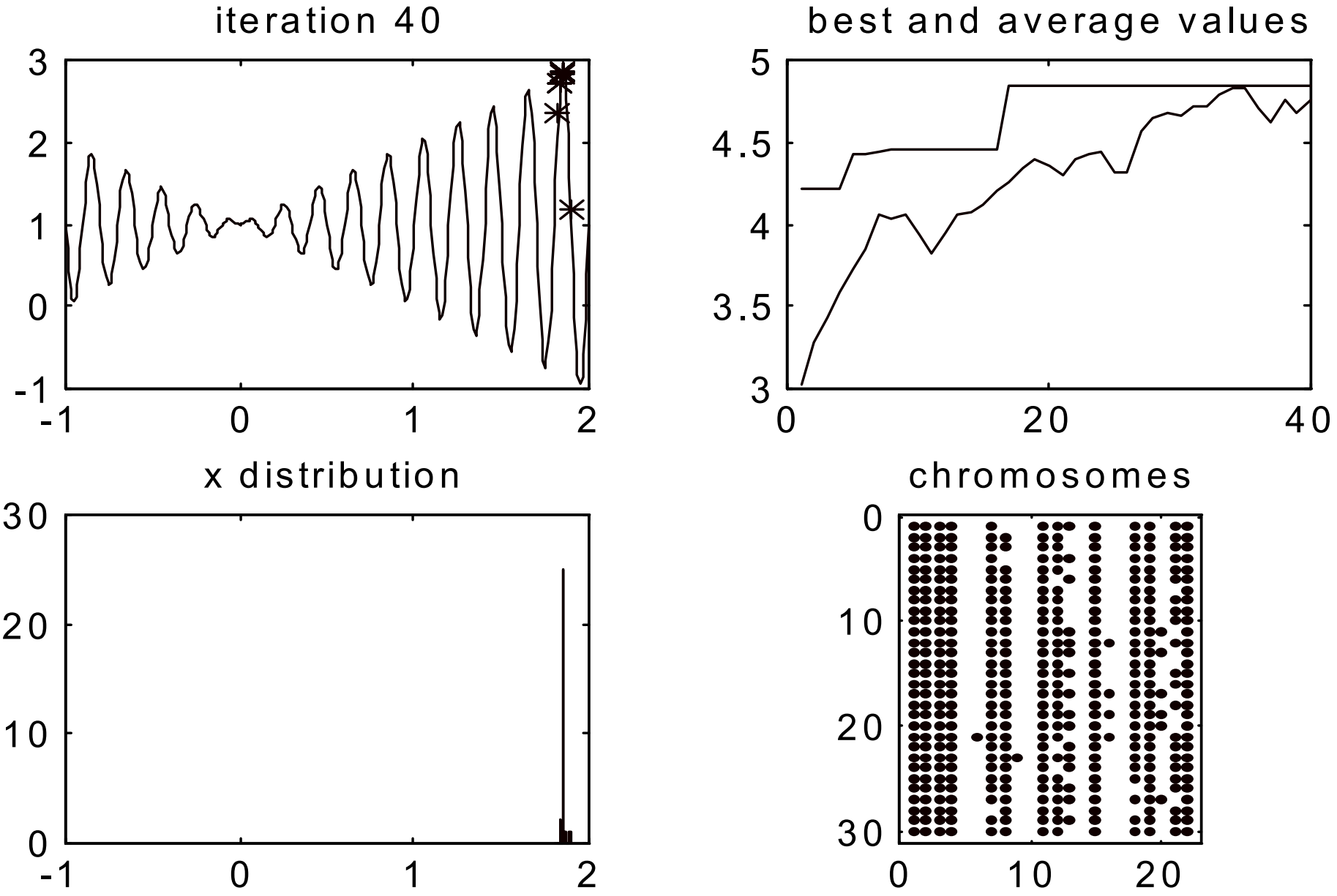
Total fitness $F = \underline{32.61}$

resulting now in an increase of overall fitness (from 32.08 to 32.61) at the end of the iteration.

The GA would now start again by computing a new roulette wheel and repeating selection, crossover, mutation and elitism; repeating this procedure for a pre-selected number of iterations.

Final results from a MATLAB GA program using parameters:

$pop\_size = 30, \quad m = 22, \quad p_c = 0.25, \quad p_m = 0.01$



Tabulated results:

x	*val*	*x*	*val*	*x*	*val*
1.8500	4.8500	1.8503	4.8502	1.8500	4.8500
1.8496	4.8495	1.8500	4.8500	1.8500	4.8500
0.3503	2.6497	1.8504	4.8502	1.8269	4.3663
1.8504	4.8502	1.8503	4.8502	1.8500	4.8500
1.8265	4.3520	1.8503	4.8502	1.8386	4.7222
1.8500	4.8500	1.8496	4.8495	1.8500	4.8500
1.8503	4.8502	1.8504	4.8502	1.8500	4.8500
1.8500	4.8500	1.8503	4.8502	1.8500	4.8500
1.8496	4.8495	1.8496	4.8495	1.8503	4.8502
1.8500	4.8500	1.8500	4.8500	1.8968	3.1880

The optimum $val = 4.8502$ at $x = 1.8504$

Hence:

$$\max_x f(x) = x\sin(10\pi x) + 1.0, \quad -1 \le x \le 2$$
$$= 2.8502 \quad \text{at} \quad x = 1.8504$$

remembering that $val(x) = f(x) + 2$